Previous Page

Tale of Protocols and Packets - an Internet Primer

By Glenn Pure

This article was prepared for the PC Users' Group magazine *Sixteen Bits*. It was published in two parts in April and May 1998.

- Introduction
- Computer networks
- Packets and collisions
- Addresses
- IP packets
- IP addresses (box)
- Addresses & domain names
- Routers & routing
- Class A, B & C addresses (box)
- Daemons in the network
- TCP
- A TCP session
- Putting it all together

Introduction

A computer is just a beige box full of complex circuit boards, wires, switches and other paraphernalia. Right?

Nope. These days it's more likely to be a beige box full of complex circuit boards, wires, switches and other paraphernalia - connected to a whole pile of other beige boxes like it! As Sun Microsystems of the US exclaims "the network is the computer".

Many of us have used networked computers for years at our workplaces. It would be hard to imagine functioning usefully any other way (except the days when the network is down!).

Nowadays, many of us are extending computer networks to our homes. Some of us connect to an employer's network so we can work after hours or take up home-based work. Lots of us are getting connected simply for play and pleasure - and to nothing less than the network of networks, the Internet. (You'll find out that 'network of networks' is actually a very apt title.)

Oh yes, the Internet again. I dare not think how many words have been written about the 'Net but this article is DIFFERENT. I'm not going to thrill you with clever things you can do on the Internet. Instead, I'm going to dabble in the standards, protocols and architecture that make it work. My aim is to avoid excessive techno-babble and explain it in a way that even the 'technically-challenged' can understand...

Computer networks

Let's dive straight in. You may have seen the acronym TCP/IP. This is the stuff of the Internet. TCP

stands for Transmission Control Protocol and IP stands for Internet Protocol. These are the two standards that work together to drive data over the 'Net. But before looking at these in any detail, we need to learn some basics of computer networks.

Figure 1 shows a simplified computer network. For comparison, a simple telephone network is shown in figure 2. Let's look at the phones first. The black lines connecting each phone to the exchange each represents a pair of wires. Every phone connected to the exchange is connected via its own separate, private pair of wires. This means that at the telephone exchange end, there are huge, thick bundles of wires going in: one pair for each subscriber. When a call is made from a phone on Exchange 1 to a phone on Exchange 2, the switches in the two exchanges connect the two relevant phone lines together via one of the trunk lines between the two exchanges. The significant thing to note is that once the connection is made, it remains unbroken for the length of the call. Both subscriber phones will be 'busy' and no other calls are possible because their lines are occupied with the current call. There is a permanent and private physical connection established between the two phones for the duration of the call.

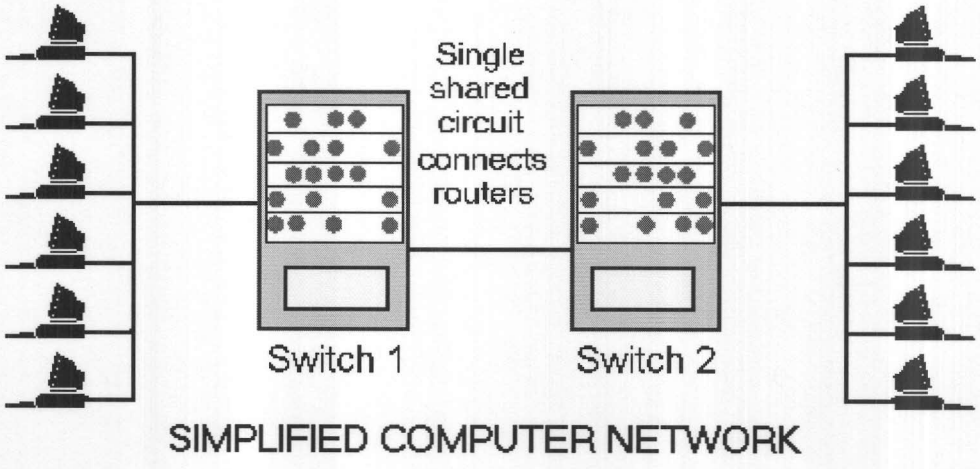


Figure 1

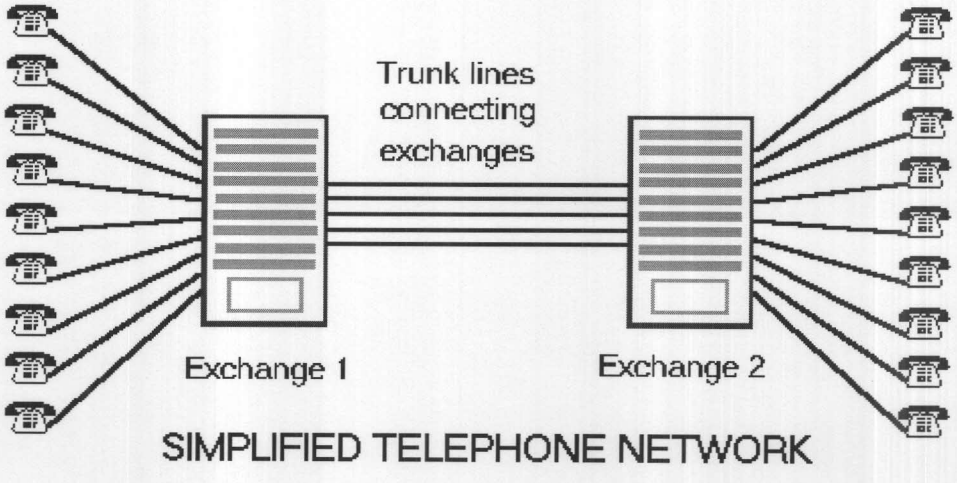


Figure 2

The computer network is very different (figure 1). Any computer connected to a particular switch is connected via a single cable which it shares with all other computers connected to that switch (computers use special 'switches' which I'll explain a bit later). There is no private line for each computer. When a 'call' is connected between a computer on Switch 1 and a computer on Switch 2,

the data first travels down the common cable to Switch 1 which shunts it onto the cable connecting the two switches. Switch 2 then switches the data onto the shared cable that goes out to the computers on the right side of the figure.

The call between the two computers might remain active for minutes, hours or days but is only allowed to occupy the cable for brief moments every now and then, when data is ready to be exchanged. All the other computers sharing the cable can have active calls over the same period and can exchange data - but they have to wait until the cable isn't tied up with another caller's data. A single computer can also have multiple 'calls' in progress at one time. It simply jumps from one to another as needed. In the case of the computer network, there are multiple virtual connections established over a single permanent physical circuit or connection (the single shared cable).

An analogy for the computer network, using the telephone example, would be a conference call among a group of people. Each person is connected to all the other parties, and can address anyone, at any time and in any order - provided no-one else is currently talking, of course. This analogy illustrates another important point which applies to the computer network: the conference callers can't conduct a private conversation with any one of the other callers because everyone else can hear. Any computer sharing the same network cable as other computers can obviously hear all the other conversations going on. This is one of the features of computer networks that hackers can take advantage of to intercept and interfere with other 'calls' (although there are solutions to this problem).

Be aware that the computer and telephone descriptions are simplifications. Also, technology is constantly on the move such that computer networks are increasingly being designed like the connections between telephones and exchanges: with each having a private cable. This enables network engineers to do nifty things like isolate computers onto different virtual networks (virtual LANs). Likewise, telephone trunk routes will, more times than not, be 'digital' and use very similar technology to that which connects two computers.

You might be wondering how this relates to your humble (or not so humble!) computer which has a connection to the Internet. Well, your computer isn't directly networked (ie shares a common cable) with other computers. But the computers at your Internet service provider are. In the case of the PCUG, these are The Internet Project (TIP) computers. Much of what I cover will describe what happens to your data once its gets into the big wide world at your ISP and beyond.

Packets and collisons

I haven't quite finished on the problems that arise when computers share a common cable. When does a particular computer know it is allowed to 'have its go' on the cable? There are many ways this can be done and it varies with the type of network. All of them rely on the sender breaking his or her data up into lots of small chunks that won't tie up the cable for very long. These chunks of data are called packets or frames. There is no standardised terminology. For the sake of this article, I will stick mostly with the term packet. Computer and digital networks are often described as packet-switched networks and the concept of packets is fundamental to such networks, including the Internet.

There are a few different ways that computers can avoid turf battles on a shared cable. While you don't really need to understand this, you might find it interesting all the same. One simple method is to give each machine a small, regular slot of time. Each computer keeps track of the time and can easily predict when its turn comes up next. The name for this technique is time division multiplexing.

It is not used very often because it suffers from a major weakness. Can you guess what that is? To give you a hint, what would happen if one computer suddenly needed to send a lot of data over the cable while the others remained fairly idle. Would it be efficient to make this busy computer wait around for a time slot while the others weren't making good use of their time slots?

A more efficient method is to allow computers to fire their packets of data willy nilly (at least initially), but if someone else happens to send their packet at the same time, then they each wait a short but random time before sending again. The chance that they will both choose the same random time is very small and another packet 'collision' can thereby be avoided. This method, in fact, forms the basis of one of the most widely used network technologies: Ethernet. Just to let you know what I am saving you from by way of gobbledegook, here is the official name for this collision detection protocol: Carrier Sense Multiple Access with Collision Detection, or CSMA/CD for short.

Addresses

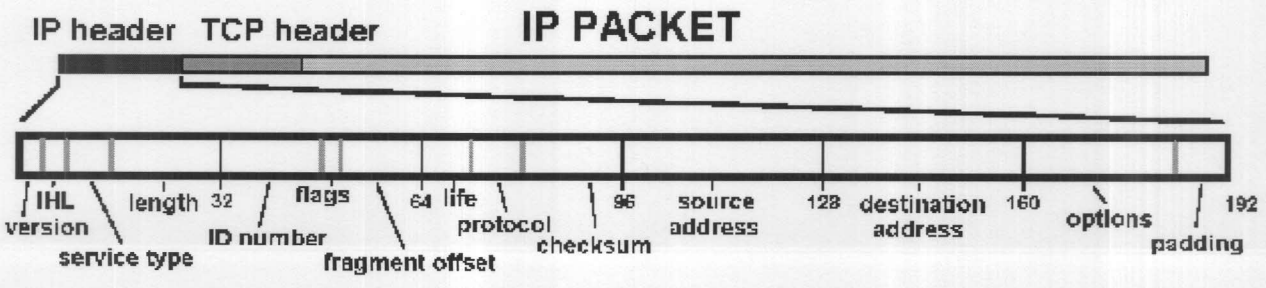
From a wiring point of view, the computer network is obviously a lot simpler (and potentially cheaper). But there is a particularly worrisome question that this architecture raises... For any data transmitted on the cable, how do the computers and the switches know where the data is coming from or going to?

The solution is conceptually very simple. An address is simply added to the start of each data packet, just as a paper envelope would have a destination address on the front. The address specifies the computer the packet is being sent to. This is where TCP/IP comes in. The 'IP' part provides the addressing feature. And just like ordinary postal mail, it's useful to have a sender's address in the packet header. IP provides this and a number of valuable features as well.

IP packets

It's now time to have a more detailed peek at what an IP packet looks like before running through an example of how it works in the real world. Figure 3 is a diagrammatic representation of an IP packet. (The official documents describing IP packets actually call them datagrams, but I'll stick with the term packet here). An IP packet is simply a long string of bits (ones and zeroes). The bits are transmitted one after the other over the network cable or phone line, starting from the left. Since transmission is from the left, the first data transmitted is the IP header. You will also notice the TCP header lurking behind it - but I won't be explaining its role until later. The remainder of the packet contains the actual sender's data - which could be anything -- maybe part of an email message, or a piece of a Web page.

Figure 3: Structure of an IP packet with an 'exploded' view of the IP header.



The IP header is the first thing to be transmitted because the network 'switches' need to retrieve the destination address before the packet can be forwarded in the correct direction. It makes sense, therefore, to put this information up front. Don't ask me, however, why the destination address was put down the tail end of the header!

As you can see, there are lots of other fields in the IP packet. A brief explanation of each will start giving you some hints about the other capabilities of IP. I have also indicated in brackets the number of bits occupied by each field:

1. **version** (4 bits): This is the IP protocol version. The current version is "4". Believe it or not, version 4 was defined way back in 1981. IP version 6 is on the drawing boards now and will probably start to be deployed in a major way in the next couple of years (there was no version 5).
2. **IHL** (4 bits): This is short for Internet Header Length and specifies how long the IP header for the current packet is. The IP packet header can actually be a variable length as we will see soon.
3. **service type** (8 bits): Contains space for a precedence code and a service quality code. The precedence code allows a packet to be specified, for example, as a "network control" packet. The second code is the service quality code and allows the sender to request special needs for example whether the packet should be transmitted with minimum delay (as opposed to 'normal' delay), or the degree to which errors can be tolerated during transmission.
4. **length** (16 bits): Contains the total number of bytes (8 bit words) in this particular packet, including all the headers. The minimum suggested size for transmission efficiency back in 1981 was 576 bytes, but I've got no idea wanted the current preferred size might be.
5. **ID number** (16 bits): Contains an arbitrary 'serial' number which helps identify the packet. The ID number is especially useful if the packet has to be broken into smaller fragments during transmission as it aids reassembly at the destination.
6. **flags** (3 bits): Contains a code which specifies whether the network is allowed to break the packet into smaller pieces during transmission. Some networks might only be able to handle shorter packets. If this flag is set to ban fragmentation and it reaches a network that can't handle it, the packet may simply be dropped in the bin. (The network can be a cruel place!)
7. **fragment offset** (13 bits): This is used only for packets that are fragmented during transmission and specifies where in the original (unfragmented) packet the bytes in this fragment are from.
8. **life** (8 bits): This is an interesting one since it starts off containing the number of seconds the packet is allowed to spend in the network before it is dumped. This value is specified by the sender. Every network switch that processes the packet must reduce the value by at least one second, even if the switch take less than a second to deal with it. In other words, this field changes as the packet moves through the Internet, and once it reaches zero, it's bye bye packet!
9. **protocol** (8 bits): Contains a code number that specifies the next header type in this packet, for example, whether the next header is a TCP header or some other type.
10. **checksum** (16 bits): Contains a checksum of the IP header to help detect whether the packet has been damaged during transmission. If it has, then the packet is unceremoniously dumped.
11. **source address** (32 bits): This contains the unique address number (IP address) assigned to the sender.
12. **destination address** (32 bits): The unique IP address of the receiver. options (variable length): Enables a whole pile of interesting things to be specified or requested like the security level of the packet (the Internet protocol was originally designed for the US defence department).

Selecting other options can force the packet to go via a specific route or ask the packet to keep track of where it had to go on the network to get to its destination. Where a route is specified or recorded, the format is simply a list of IP addresses of each network switch that the packet went through.

13. **padding** (variable): All IP headers must finish on 32 bit boundaries (in other words, the number of bits in the IP header must be divisible by 32). Padding, consisting of zeroes, must be added to bring the header up to the next 32 bit boundary.

There are a couple of fields above worth a further comment or two. One of these is packet length. The size of packets is a tricky thing to specify. Because the IP and TCP headers might occupy, between them, almost 400 bits, they add a lot of 'overhead' to the packet. Ideally the ratio of data bits to header bits should be maximised to keep this overhead down. However, networks physically can't transport packets with no chance of error. And the larger the packet gets, the more chance that it will score some damage during transmission and must be discarded. So the packet length is something of a compromise between network error rates and packet header overhead.

The IP source and destination address fields are probably the most interesting... and the most important to this article (see box on IP addresses). Every computer or device connected to the Internet has its own 32 bit address. This means there are 4,294,967,296 possible addresses ie just over 4 billion. Like the 640K limit on RAM in the original IBM computer, no one ever thought this would be exceeded. And, like IBM memory, there are now serious concerns about running out of 32 bit IP addresses. IP version 6 (the next version) will solve this, at least for a 'little while'. IPv6 addresses are 128 bits long. This means there are 2^{128} possible addresses. To put a number of this size in context, the total number of atoms in the Earth is estimated to be 2^{170}.

IP addresses

IP addresses are written in an unusual way. They consist of four numbers separated by dots, for example: 10.157.23.98 The dots are just convenient separators. Each of the numbers represents one of the four bytes that make up the 32 bit address, and each byte has a possible value of 0 to 255.

Every IP address has to be different to ensure that every user or machine on the Internet can be uniquely identified. The only effective way to achieve this is to leave a single body in charge of all IP addresses. There is such a body and it is called the Internet Assigned Numbers Authority (IANA). However, rather than issue all the IP addresses itself, it delegates the job to recognised authorities around the world, and issues each of these with a block of addresses that can be handed out. There are four such bodies in Australia: Melbourne IT, Netlink Holdings, Capital Networks [in Canberra!] and Moniker. (The politics of IP address allocation are very complex and warrant a whole article in themselves.)

These bodies also register names associated with the IP addresses, simply so we don't have to remember a 32 bit number to get to a site or mailbox on the Internet. Instead, we remember the name associated with that address, for example, www.pcug.org.au instead of 203.10.76.34. (By the way, this 32 bit number should work if you try it, but I recommend you

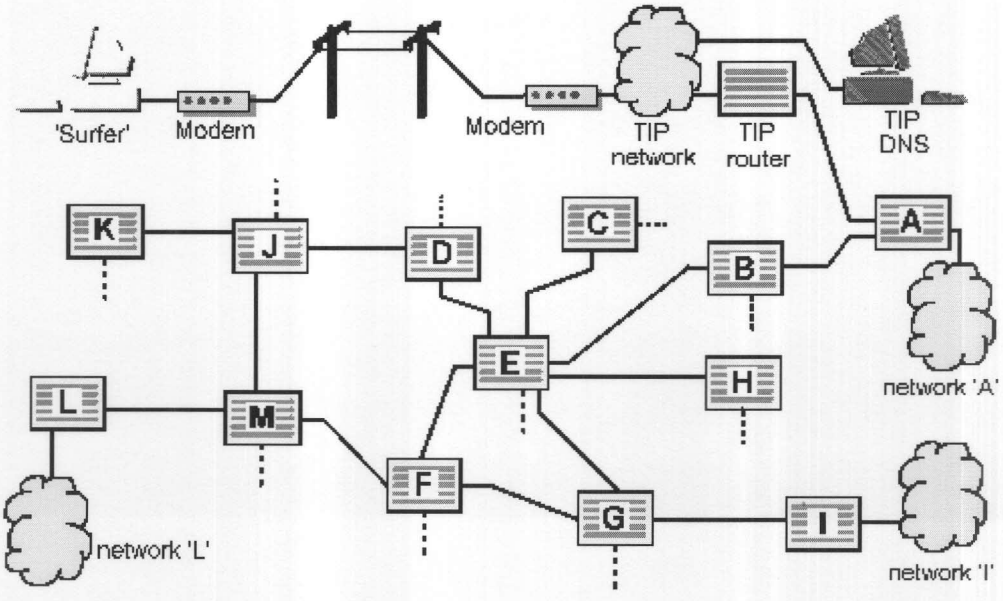
don't as there is no guarantee that TIP won't be reconfigured by our TIP administrators tomorrow, and the address changed.)

Just as we need telephone directories to find people's telephone numbers, the Internet has to provide directories so that the IP addresses of Internet sites can be found. These 'directories' are stored in computers on the 'Net and form part of the Domain Name System (DNS for short).

Addresses and domain names

Now it's time to look at what actually happens when you send or receive data between your computer and the Internet. As an example, let's look at what happens when you fire up your World Wide Web browser to take a peak at a Web site of interest. Figure 4 shows a layout of the hypothetical section of the Internet (the real thing is similar but vastly bigger than this). You will need to look at this diagram to make sense of the explanation below.

Figure 4: A hypothetical section of the Internet showing a 'surfer' connected via modem to TIP (top part of figure). The shaded boxes are network switches called routers. The routers each have a private network connected to them, shown either as a cloud or abbreviated as a dotted line. The routers could be scattered around the world and are connected to one another by public communication lines such as satellite links, undersea cables, optical fibres etc which are rented off phone companies or the like.



At the moment, you only know the name of the site. You have no idea what its IP address is. So the first thing to do is look up the IP address. To get this process underway, an IP packet is created on your computer. The data section of the packet (not the header) will contain the name of the site you want to look at (for example www.intel.com). The IP header of this packet will contain, as its destination address, the IP address of your preferred Domain Name System (DNS) server. This server looks up "www.intel.com" to find the corresponding 32 bit IP address. It then creates a new IP packet containing this information and sends it back to you by putting your IP address as the destination in the IP header. We'll look at the process by which the packet gets routed to and from you in a minute since it is the same for other packets.

Before finishing on the topic of the DNS, there are a couple of interesting questions to answer. How do you know which DNS server to send your request to in the first place? Well, this is the only address where you must tell your computer the numerical IP address. When you first subscribe to TIP or another Internet Service Provider (ISP), you will be told the local DNS server's IP address (or it will be buried somewhere in the start up kit your ISP gives you and automatically loaded for you). In the case of TIP, it actually has its own DNS (the IP address is 203.10.76.34).

What happens if you make an error in the name of the site you send off to the DNS server? Try it some time and have a look at the message on your browser screen. It will say something like "DNS name lookup failure... The named host probably does not exist.". Hopefully you'll now have a better idea of what that message means.

Our browser now has the 32 bit address for the Web site. Another packet is created containing this as the destination address in the IP header. The data section of the packet will contain a message requesting the Web page. (The format for this message is not of interest here; it is part of the HyperText Transfer Protocol, or http for short.) We're interested in how the packet gets to its destination and how the requested information is returned in response.

Back to figure 4 again. Let's assume we are fetching a Web page from the Web server located on network 'I' (the Web server is nothing more than a computer on this network running Web server software).

Routers and routing

The first thing that happens is the packet is turned into a stream of bits by your computer and transmitted to TIP over the phone lines, via a modem at each end. Once in the TIP network, it will be one of many packets on the shared cable that connects all the machines at TIP, including the TIP router. Router is the term used for a network packet 'switch' and they are just specialised computers (there are a few other switch types, but we won't get into these). The router's job is to look at the destination address in each packet and decide whether it should be sent to the outside world on one of its ports, or just ignored. In the case of the packet we have just sent to TIP requesting a Web page, the TIP router reads the destination address and says to itself "Aha, this packet is not addressed to any IP address in TIP. It therefore must be sent forth into the wider Internet."

In figure 4, it happens that the TIP router is only connected to one point on the Internet: router A (all of the boxes with letters in them in figure 4 are routers). Note that there is also a 'private' network (network 'A', shown as a cloud) connected to router A. In fact, the only reason router A probably exists is because the organisation that created network 'A' decided it wanted an Internet connection and so put the router in place to connect itself. The managers of the TIP network and network 'A' would have to agree with each other before renting a line to connect them. Network 'A' could be just about anything including a private network in a company, another organisation like TIP, a university, a public Internet Service Provider (ISP) and so on.

(All the other routers shown in figure 4 will probably have private networks connected to them and network clouds are shown for some. To save space, the networks connected to other routers have simply been abbreviated as dotted lines. The Internet is sometimes referred to as a 'network of networks'. The logic behind this name is that the routers are connected to one another in one huge network while there are smaller private networks connected to each router.)

Coming back to our packet, from here on the same thing happens each time it gets to another router. The destination address is examined by the router to first determine whether or not it is addressed to any machine on that router's private network -- if not, it sends it to the next router. The packet will finally get to the router connected to the destination network, which in our case is network 'I'. Once it reached router I, that router would look at the destination address and say to itself "Wow, this packet is actually addressed to someone on my network. How exciting. I'll forward it on to my local network straight away" (routers have pretty boring lives).

The tricky part with routing comes when there is more than one router to send the packet to (for example, router E). But let's first backtrack and look at what happens to our packet at router A. This router could, in fact, send the packet to router B or back to the TIP router if it wanted. To make the right decision (ie send it to router B), router A not only has to know what IP addresses exist on network 'A' (so it can decide whether or not to forward any packet on to network 'A'), it must also have some idea of the network addresses that are connected to its other ports. On the TIP side, that will be the relatively short list of TIP addresses. On the router B side, all the remaining IP addresses on the Internet are found. The simplest way for router A to deal with this is to record only the addresses that are connected to the TIP port and to private network 'A' in its routing table. Anything that doesn't match these will simply be forwarded to router B (the router's equivalent of buck-passing!). In this situation, router B would be referred to as the default router for router A.

The job of configuring router A is obviously fairly straightforward and can probably be done manually by the administrator of network 'A'.

Routing made simpler: class A, B and C addresses

There are three classes of IP addresses that are issued to end users: A, B and C.

Remember that IP addresses consist of four bytes, each with a range of values from 0 to 255. When all but the last byte is specified in an address, this is said to be a Class C allocation. For example, 203.10.76.* is Class C. The person or organisation which is assigned this can use all of the 256 possible values for the final byte to assign to people or machines on their network (don't forget that anyone who wants to use the Internet has to have their own unique IP address).

A Class B allocation only specifies the first two bytes, for example 132.10.*.*. There are 65,536 possible values for the remaining two bytes. Such large amounts of address space would be reserved for either very large organisations.

You have probably guessed that Class A allocations only specify the first byte, for example 45.*.*.*. Each Class A allocation can provide almost 17 million addresses. A Class A allocation would be sufficient for a whole country in many cases.

Because addresses have mostly been allocated in blocks as just described, routers can save themselves a lot of work. Rather than storing

all the 4 bytes addresses which are accessible on each port (after all, there are 4 billion addresses total), they only need to store the class A, B or C numbers in their routing table. This vastly reduces the storage and processing power required in the router down to a practical level. An analogy in the ordinary postal system is the use of street, suburb and country identifiers to help the postal service to efficiently sort and dispatch mail. For example, a postal worker sorting international mail only needs to look at the country in the address. He or she doesn't have to worry about the other parts of the address.

Router E, on the other hand, can't get away this easily. It has to store a lot more information in its routing table because it has many more active ports with routers connected. (It will probably be a lot busier too, and consequently have a faster processor and bigger memory.) Router E can still have a default router, but that doesn't greatly diminish the task of knowing what IP addresses are connected to its other ports.

'Daemons' in the network

The job of manually entering all the routing information that router E needs is a bit excessive although some manual configuration will always be essential. Router E's task is compounded by the fact that the network is constantly changing to a greater or lesser degree. Fortunately, there are automated ways for it to find out about the addresses that are accessible on its various ports. Most routers come with routing daemons. These are software routines that create and exchange packets with other routers solely to swap routing table information.

Daemons also have other uses. Let's look again at our packet being sent from TIP to the Web server on network 'I'. Once this packet gets to router E, it has a few choices about where to go next. It could go directly to router G. But maybe the line between E and G is slow or error prone. Hence, it might be preferable to send the packet via router F.

Now what happens if router F fails? Will router E find out about this? This is where the daemons come in again since they also help detect failures in the network. If a router somewhere goes off line or crashes (for example, router F), it will stop sending its routing information. Other routers listening for this information (like router E) will suspect a problem and will, after a short time -maybe a minute - delete all of F's routing information from their routing tables. It's as if router F didn't even exist. Router E will still have the information from router G which tells it there is a path to network I through G. Hence, router E will forward the packet this way instead.

So the next time you have trouble reaching an address on the Internet, you might rightly blame the "daemons in the network"!

Jokes aside, the Internet is actually a very robust network. One of the key reasons is that there are usually multiple paths to the same destination. The Internet protocol was specifically designed to take advantage of this so that even if parts of the Internet were out of action, the packet would probably get through. After all, the Internet protocol was designed for the US defence department which was conscious that an enemy might be able to take out part of a network, but not all of it. In those circumstance, they would still be able to communicate over much of the network.

TCP

Time to look again to our original request to fetch a Web page from network 'I'. Our packet has finally reached its destination. The Web server on network 'I' will intercept it and respond with a copy of the requested page. This will probably be large enough that many packets will be needed to send it back to us. The Web server will send these packets off in the correct order, but there is no guarantee they will reach our end in the same order because it is quite possible for each packet to go by a different route and thereby take a different amount of time in the process. This is where the Transmission Control Protocol (TCP) comes to the rescue...

The Internet protocol is responsible for ensuring that the packet gets from the sender to the destination. IP doesn't care if there is one or one hundred packets that need to be sent. It is worries about each individual packet when it gets it. TCP, on the other hand, is responsible for ensuring that the stream of data which may have been broken up into many IP packets, actually gets to the receiving end without any parts damaged or missing, and in a form that can be reconstructed easily.

Figure 5: The TCP header in an IP packet

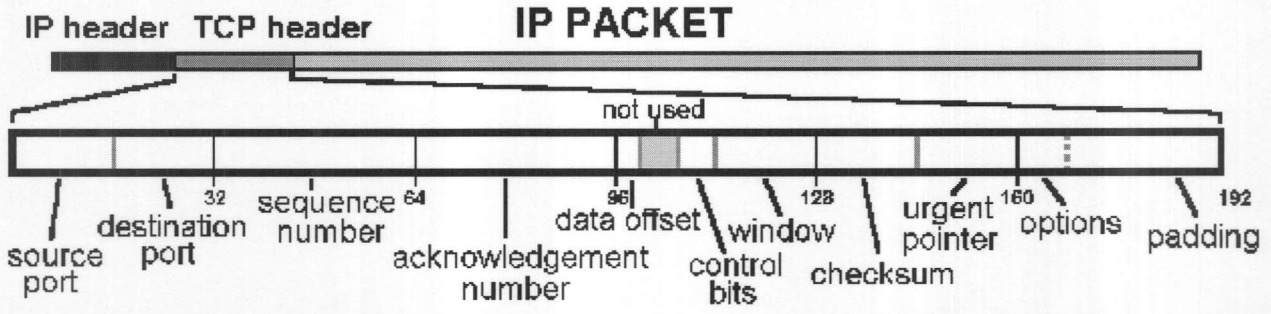


Figure 5 shows the structure of the TCP header. Here's a brief rundown on what each of the fields in the header does:

1. **source port & destination port** (16 bits each): Don't confuse these with router ports. They have nothing to do with routers. The port fields are numbers. They help the receiver to work out what to do with the data in the packet. Some packets may be addressed to an email application, others might be interacting with a Web page service. A single machine with one IP address may be hosting these applications and a whole heap of others (this is what happens in TIP). To distinguish between them, they are each given a different port number. There is actually a standard list of port numbers published by the Internet Assigned Numbers Authority (IANA) to cover most of the common applications.
2. **sequence number** (32 bits): This is probably the most important field. It contains a number which is used to identify where in the original data stream the data in the current packet came from. Say we wanted to send 20,000 bytes of data to a destination. Let's also assume the data is broken up into 20 IP packets for transmission, with each packet containing 1,000 bytes of data. TCP first assigns an arbitrary number to the first of our 20,000 bytes. For simplicity, we'll call this number 'z'. The sequence number of the first packet will be 'z'. The sequence number of the second packet will equal ('z' + 1,000), the third packet will be ('z' + 2,000) and so on. The sequence numbers allow the data to be reassembled in the proper order at the receiving end.
3. **acknowledgment number** (32 bits): TCP is quite a bit different from IP in that it involves

two-way communication between the sender and the receiver. One role for this is the acknowledgment of packets received. When a packet is successfully received undamaged at the destination, the receiver sends an 'acknowledgment' packet back to the sender. This packet contains a running total, in the acknowledgment number field, of how much data has been successfully received at the destination so far.

4. **data offset** (4 bits): The size of the TCP header (expressed as the total number of bits in the header divided by 32)
5. control bits (6 bits): This field contains status or command information, for example whether the sender has finished transmitting data and wishes to close the call, whether there is urgent data in the current packet or whether this packet is an 'acknowledgment' packet (see explanation of acknowledgment number above).
6. **window** (16 bits): This field only has meaning in an 'acknowledgment' packet. It is sent from the receiver to the sender and indicates the number of further data bytes which the receiver is willing to accept at this time. Back in 1981 when this protocol was specified, computers and memory were very expensive. If these computers were busy, they might be swamped with more data than they could store and handle. The window was designed as a way of politely requesting that the sender limit how much was sent before it got an acknowledgment.
7. **checksum** (16 bits): This contains a checksum calculated for the entire contents of the packet including the TCP header, the IP header and the data. If the packet is damaged during transmission, the checksum calculated by the receiver will not match and the packet will be discarded.
8. **urgent pointer** (16 bits): If the 'urgent' status has been flagged in the 'control bits' field (see above), the urgent pointer will identify the location of the urgent data in the packet - otherwise this field is ignored.
9. **options** (8 or 32 bits): This field is either one byte or four bytes long, depending on the option. There are only three possible options, and the only one worth mentioning is the 'maximum segment size' option. This is used when a 'conversation' between two IP stations is being negotiated. The receiver uses this field to tell the sender the maximum packet size it will accept.
10. **padding** (24 or 0 bits): Length of this field depends the length of the options field. Zeroes are used for padding. The padding is solely done to make sure the TCP header finishes on a 32 bit boundary (that is, the number of bits in the TCP header is divisible by 32).

A TCP session...

TCP is quite a bit more complex than IP because it is designed to conduct a two-way conversation between the sender and the receiver. The role of some of the fields above might become a little clearer once you know how this conversation occurs. There are several ways a conversation can start. In the example below, the sender initiates the conversation. Here's an outline of what occurs, with some steps removed for simplicity (remember, all of these messages are 'wrapped' inside properly addressed IP packets):

1. The sender creates a special TCP message called a synchronise message, or SYN for short. The control bits field in the TCP header is used to specify that the current packet is a SYN. The packet contains no data but it does contain the starting sequence number for the data that is to be transmitted.
2. If the destination machine is in the receiving state, it will acknowledge the SYN message by returning a packet with the sender's starting sequence number in the acknowledgment number field.

3. For various reasons, the sender needs to acknowledge this message. Once this happens the connection between the sender and the receiver is said to be established.
4. Data transmission now occurs. The sender transmits the first chunk of data. The TCP header will contain the starting sequence number for the data.
5. The destination will receive the packet and send an 'acknowledgment' packet back to the sender to indicate success. Steps 4 and 5 are repeated until all of the data is transmitted. It is possible, however, for some of the packets to get lost or damaged. In these cases, the receiver will dump any damaged packets then sit and wait. Meanwhile, back at the sender, a clock is ticking away. Once this clock reaches zero (probably after a few seconds), the sender will retransmit any packets previously transmitted but which have not yet been acknowledged. The TCP/IP stack at the sender's end keeps copies of all unacknowledged packets specifically for this purpose. Once an acknowledgment has been received for a packet, it is deleted from the sender's stack.
6. When all the data has been sent, it's time to close the connection. There are several ways to do this. In this example, the sender will initiate the 'close'. The sender creates a 'finished' packet, or FIN for short and sends this to the receiver.
7. The receiver acknowledges the sender's FIN. The receiver might still be waiting for data packets in which case it will wait until all have been received and acknowledge them. It will then send its own FIN packet to the sender.
8. Finally, the sender will acknowledge the receiver's FIN and the connection will enter the closed state.

Putting it all together

The Internet protocol and the transmission control protocol work hand-in-hand to ensure that data gets from the sender to the receiver reliably and in an uncorrupted form. IP is responsible for creating packets and for addressing them to the correct destination. IP is mostly worried about the network and shifting packets around. TCP, on the other hand, provides crucial quality control. It constantly checks up on IP to see that its packets and getting through. TCP ensures that data is split up properly for transmission (by IP), and is reassembled correctly at the destination end.

I have simplified the story somewhat and left out a few of the more estoric details. There is also lots about computer networking that I haven't covered like the layer structure of the OSI model. But this article was only intended as an introduction, although one with enough detail to satisfy the curious... and it has already turned out far longer than I anticipated.

If you want to read more about TCP/IP and computer networking, there are heaps of resources on the 'Net. Try a search with your favourite search engine. One source of information that is extremely valuable is the RFC documents. 'RFC' is short for 'request for comment'. RFCs are the technical documents that specify all of the Internet protocols and standards. Be warned though, there are thousands of them. Much of this article was drawn from the IP and the TCP RFCs (RFC 791 and RFC 793 respectively). The RFCs are prepared by individuals and working groups of the Internet Engineering Task Force (IETF) and are available on many sites. Try this one to get you started: http://www.nexor.com/public/rfc/index/rfc.html